University POLITEHNICA of Bucharest
Faculty of Automatic Control and Computers
Computer Science and Engineering Department

Diploma Thesis

Low-power communication protocol for Wireless Sensor Networks

by

Tudor Vișan

Supervisor: As. Drd. Ing. Andrei Voinescu

Bucharest, September 2014

Contents

Abstract

This paper proposes a new communication protocol for Wireless Sensor Networks designed to take advantage of alternative energy sources. Despite improvements in both energy efficiency and sensing capabilities Wireless Sensor Networks are still limited because of the limited battery life of their nodes. Similar protocols focus on low energy consumption to the detriment of bandwidth and network responsiveness and never take into account the actual environment around the nodes. Our approach is based on harvesting and storing energy from the surrounding environment is order to enable longer and more intense periods of activity while preserving battery life. . Through this technique data loads within Wireless Sensor Networks may increase thus allowing more precise and complex measurements and greater network size, all while preserving node lifespans.

Keywords Wireless Sensor Networks, alternative energy, energy harvesting, scheduling, synchronization

Acknowledgements

Mulțumiri

Chapter 1

Introduction

Chapter 2

Related Work

Chapter 3

Hardware Platform

Voi folosi o super-referință S-MAC[1] și un rahat [2]

Chapter 4

Software Platform

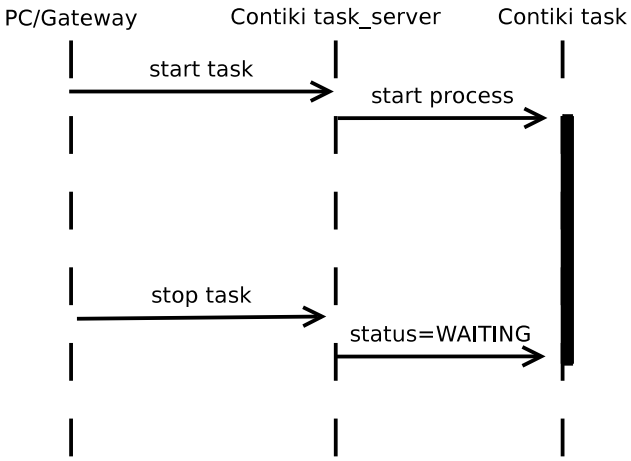

Figure 4.1: *The exchange of messages while starting/stopping tasks*

Listing 4.1: Task server snippet

```
1  PROCESS_THREAD(task_server_process, ev, data)
2  {
3      PROCESS_BEGIN();
4
5      list_init(task_list);
6
7      list_add(task_list,&el_monitor_process);
8      list_add(task_list,&el_delay_process);
9      list_add(task_list,&el_temperature_sensing);
10
```

```
11      tcp_listen(HTONS(1010));
12
13      while(1)
14      {
15          PROCESS_WAIT_EVENT_UNTIL(ev == tcpip_event);
16
17          if(uip_connected())
18          {
19              PSOCK_INIT(&ps, buffer, sizeof(buffer));
20
21              while(!(uip_aborted() || uip_closed()
22                  || uip_timedout()))
23              {
24                  PROCESS_WAIT_EVENT_UNTIL
25                        (ev == tcpip_event);
26                  handle_connection(&ps);
27              }
28          }
29      }
30      PROCESS_END();
31  }
```

Chapter 5

Other Chapters, TBD

Chapter 6

Conclusions and Future Work

Bibliography

[1] Jonathan W. Hui and David E. Culler. Ip is dead, long live ip for wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 15–28, New York, NY, USA, 2008. ACM. [cited at p. 5]

[2] Gideon Rahat and Reuven Y Hazan. Candidate selection methods an analytical framework. *Party Politics*, 7(3):297–322, 2001. [cited at p. 5]

Appendix A

Contiki API

A.1 Process macros

- `PROCESS_THREAD (name, ev, data )` - Define the body of a process. This macro is used to define the body (protothread) of a process. The process is called whenever an event occurs in the system, A process always starts with the PROCESS_BEGIN() macro and end with the PROCESS_END() macro.
- `PROCESS_BEGIN ()` - Define the beginning of a process.
- `PROCESS_END ()` - Define the end of a process.
- `PROCESS_YIELD ()` - Yields the currently running process
- `PROCESS_WAIT_EVENT_UNTIL (c)` - Wait for an event to be posted to the process, with an extra condition. This macro is very similar to PROCESS_WAIT_EVENT() in that it blocks the currently running process until the process receives an event. But PROCESS_WAIT_EVENT_UNTIL() takes an extra condition which must be true for the process to continue.
- `PROCESS_PAUSE` - Yield the process for a short while. This macro yields the currently running process for a short while, thus letting other processes run before the process continues.

A.2 uIP functions

- `PSOCK_INIT (psock, buffer, buffersize)` - Initializes a proto-socket. This macro initializes a protosocket and must be called before the protosocket is used. The initialization also specifies the input buffer for the protosocket.

- `PSOCK_SEND (psock, data, datalen)` - Send data. This macro sends data over a protosocket. The protosocket protothread blocks until all data has been sent and is known to have been received by the remote end of the TCP connection.
- `PSOCK_READBUF (psock)` - Read data until the buffer is full. This macro will block waiting for data and read the data into the input buffer specified with the call to PSOCK_INIT(). Data is read until the buffer is full..
- `CCIF process_event_t tcpip_event` - The uIP event. This event is posted to a process whenever a uIP event has occurred.
- `CCIF void tcp_listen (u16_t port)` - Open a TCP port. This function opens a TCP port for listening. When a TCP connection request occurs for the port, the process will be sent a tcpip_event with the new connection request.
- `struct uip_conn *tcp_connect(uipipaddr_t *ripaddr,u16 port, void *appstate)` - This function opens a TCP connection to the specified port at the host specified with an IP address. Additionally, an opaque pointer can be attached to the connection. This pointer will be sent together with uIP events to the process.
- `uip_connected()` - Has the connection just been connected?
- `uip_closed()` - Has the connection been closed by the other end?
- `uip_aborted()` - Has the connection been aborted by the other end?
- `uip_timedout()` - Has the connection timed out?
- `uip_newdata()` - Is new incoming data available?
- `uip_close()` - Close the current connection.

Appendix B

Node capabilities

Task	AVR Raven™	Sparrow	Sparrow Power
Temperature sensing	✔	✔	
Humidity sensing		✔	
Voltage & Current sensing			✔
Event detection	✔	✔	✔
Alarm beep	✔		
LED signal	✔	✔	

Table B.1: Node capabilities

List of Figures

List of Tables

Listings